

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR U.S. LETTERS PATENT

Title:

**MEMORY ACCESS METHOD BY REFERENCING DATA,
AND DATA PROCESSING DEVICE USING THE SAME**

Inventors:

**Motoyuki KATO
Hiroyuki YANAGI
Shinji NAKAGAWA
Yosuke BABA**

Dickstein Shapiro Morin
& Oshinsky LLP
2101 L Street NW
Washington, DC 20037-1526

MEMORY ACCESS METHOD BY REFERENCING DATA, AND DATA PROCESSING DEVICE USING THE SAME

Field Of The Invention

This invention concerns a scheme to execute a program through the use of an interpreter language and an information processing device which contains such a program. More specifically, this invention concerns a technique to execute a program which requires that destinations to be accessed in the memory be specified by referencing data.

Background Of The Invention

Java®, the object-oriented programming language developed by Sun Microsystems, operates in an environment which has a Java interpreter to convert Java commands into byte code without having to rely on a platform. It has received a great deal of attention as a language which is well-suited to set values in information processing systems contained in various devices.

To prevent improper access to the memory, information processing systems using Java do not allow the program to directly specify a destination to be accessed in the memory, but instead specify the destination indirectly through data representing its name.

For example, a program indicating that a value should be set in a variable (i.e., in a field) or that this value be read out would designate the variable by the name of the class to which it belongs, the name of the variable and the name of the type of variable (e.g., int or byte). Programs that access methods designate the method they wish to access by the name of the class to which the method belongs and the name of the method. Generally, the processing entailed in using a given convention (such as character data indicating a name) to find the location where data are stored concerning something to be processed (a variable, a method, a

class, etc.) is called "referencing." Indicating that data are to be referenced is called a "reference request." (For example, referencing a variable, a method or a class is called a "field reference," a "method reference" or a "class reference.")

5 When executing a program which requests a reference using a given name as data, the Java interpreter which is called a "virtual machine" will retrieve the specified reference table (as will be explained in detail shortly) by means of the data represented by the name and specify the address where the data to be referenced are stored. Referencing data designated in this way and indicating the location where they are stored is called "resolving a reference."

10 Figure 6 shows the compiled result of a program which uses Java. In the figure, each dotted rectangle is one unit's worth of program code. Each code contains multiple data points. In the example, individual data are shown in the white boxes.

15 In Figure 6, 10 is the byte code program obtained by converting the message indicating that a value should be assigned to the variable. (In assembler code this is represented as "putstatic Sample.value:int".) It consists of instruction code representing the content of the request (B3) and an operand representing the address (00 06) where the content which the instruction indicated was to be referenced can be found. (Hereafter, an object program with an instruction code attached will be referred to as an "instruction.")

20 Reference data configured as shown on the right in the figure are linked to instruction 10. These data consist of a combination of various lengths of code data, to the head of which is attached a tag representing the type of data. An identification number called the "constant pool entry number" (hereafter, called simply the "entry number") is attached to each set of
25 code data, and a link is set up for each entry number. The first entry number for reference data, 06, is written into the aforesaid operand. By following the links between the various sets of code data in order, starting from the first entry, we can obtain a character string representing the name of the class, the variable, and the type of variable that were described to the
30 source program.

These reference data are stored in class units. (Hereafter, reference data in class units are referred to as "class file data.") To simplify the explanation, in Figure 6 the reference data which have been selected are for a single instruction. Entry numbers not shown in the figure are used for the reference data for other instructions.

In the figure, character data representing the class name "Sample" are recorded in entry number 20; data representing the variable name "Value" are recorded in entry number 27. The character string "I", which represents int, the name of the type of variable, is recorded in entry number 17. The character strings of data in entry numbers 20, 27 and 17 are encoded. Code data representing the number of bytes in the character string and the tag "01", which indicates that character strings of encoded data are recorded, are added in front of the encoded data.

The data in entry numbers 1, 6 and 9 indicate what location the aforesaid character string is to reference.

The data for entry number 6 have the tag "09", which means "field reference". This tag indicates that the following data are field reference data. Code data representing the entry numbers 1 and 9 are recorded in entry number 6.

The tag "07", which means "class reference", is attached to entry number 1. This sets up a link in the character data which represent the aforesaid class name to connect this entry number with entry number 20, the destination to be referenced. The tag "0C", which means "name and type reference", is attached to entry number 9. This links entry number 9 to entry number 27, where the character data for the aforesaid variable name are stored, and entry number 17, where the name of the type of variable is stored.

This configuration of data allows us to follow the links in the reference data in an orderly fashion beginning with entry number 6, which is written into the operand of instruction

10. In this way we can obtain the names of the class, the variable and the type, which we need for a field reference. Other commands express the data in the same fashion. When the source program is compiled, instructions are generated which serve as links to the reference data that represent the actual content of the designated reference.

5

Reference data can be shared among a number of instructions which all require the same reference. For example, let us assume that the aforesaid instruction 10 occurs in a number of places in the program, and that there is another instruction (for example, an instruction obtained from the program "getstatic Sample.Value:int") which requires the same field reference as instruction 10. The aforesaid entry number 6 is then written into the operand of each of these instructions to set up a link to the reference data in the aforesaid Figure 6.

10

When the interpreter executes each instruction, it recognizes the content of the command from the aforesaid instruction. It follows the links in order starting from the entry number written into the operand, and it finds the content of the designated reference and the data which represent the name used in the reference.

15

For the different variables and methods set up in each class, a reference table is created in the system to specify from the name data the location in the memory where those variables and programs are stored. The interpreter uses the name data it obtains from the aforesaid reference data to look up the aforesaid reference table. It executes the necessary processing to find the destination designated by the aforesaid instruction.

20

Figure 7 shows the configuration of the table of classes and the table of fields (11 and 12 in the drawing) used to find field references. In Figure 7, 13 is a heap memory which serves as the work area for all the objects designated by the program. 14 is a table where character strings such as names of classes or variables are stored.

25

As the aforesaid class file data are generated, data concerning the class are stored in the aforesaid tables 11 and 12. Data stored in class table 11 include the top address of the class;

30

the top address of the string; the name index for the class itself; the name index for the parent class (the super index); the size of objects in the class; and access flags. (Hereafter, these data are all called "class data".)

5 The top address of a class is the address where the head data of the aforesaid class file are stored. The top address of the string is the head address of the area in table 14 where character strings such as the names of classes and variables for each class are stored. The name index consists of the constant pool entry numbers which indicate the names of the classes in the aforesaid reference data. Access flags indicate whether or not a class can be accessed.

10 An individual index (hereafter referred to as a "class table index") is attached to the class data for each class.

15 For every variable the program creates, various data are stored in the field table, including an object offset; an index to the class table; a name index; a type index; and access flags. (Hereafter, these data are collectively referred to as "field data".)

20 The aforesaid object offset is a value indicating the relative location of the field assigned to the aforesaid variable in the work area for the object which contains that variable. This value is determined when the class file data are created. The size of the work area is determined in which the instance of each class must be stored in a heap memory, and a storage area is allotted for each variable in this work area.

 However, values for static variables are themselves stored in the field table.

25 The index to a class table is attached to the class data in the aforesaid class table for the class which contains the aforesaid variable. Name and type indices are constant pool entry numbers for the data which represent variable and type names in the aforesaid reference data. Access flags, as was just as was described above, indicate whether or not a variable can be accessed.

A separate index (Hereafter referred to as a "field table index") is attached to each set of file data.

To find a field reference, the interpreter uses the name of the class it obtains from the reference data linked to the aforesaid instruction to retrieve the class table and get the table index for that class. It then retrieves the field table and extracts from the data containing the aforesaid index the field data whose variable and type names match the names it got from the aforesaid reference data. It gets the address of the storage area assigned to the variable from the object offset value in the field data.

If the interpreter is to execute the same instruction again, when it obtains the result of referencing the aforesaid field, it rewrites the instruction code from the aforesaid instruction into the code which it has found. It rewrites the operand into the number of the field table index it obtained from the aforesaid field reference. As a result of this rewriting, when the interpreter executes the instruction, it can immediately read the object offset that is in the field table from the field table index written in the operand and thereby obtain the location where the designated variable is stored.

To aid in referencing methods, a reference table (the method table) is set up to direct the interpreter, via class and method names, to the locations where programs for methods are stored. The interpreter obtains the index for the method table from the instruction and from the name of the method it reads out of the reference data linked to the instruction, and in this way it references the method. If it is to execute the instruction again, the interpreter rewrites its operand in the aforesaid index for the method table which it has obtained.

Figure 8 shows the chain of processing in a system using a byte code program generated by the aforesaid Java source program, from the time the system is started until the program is completed.

The object program and class file data constituting this system are stored in an internal ROM. When power is supplied to start the system, the object program is loaded into a RAM, and the class which is to be activated first by the start-up program is indicated (Step 1). In Step 2, the class file data for the indicated class are read out into the RAM. In Step 3, the data for the aforesaid class whose file data have been read out are stored in various reference tables, including the class table, the field table and the method table.

In Step 4, when the environment for executing an instruction has been arranged in this way, the first instruction is read. If this instruction requires that a class be referenced for which class file data have not yet been created, we move from Step 5 to Step 6 and designate that file data be created for this class. In response, file data are created and entered into the reference tables in Steps 2 and 3. When this has been completed, the aforesaid instruction is read again.

If file data already exist for the class whose reference is required by the instruction, the interpreter finds the names of the class, variable and method in the file data. If this instruction is the first one executed after the system is started up, its instruction code will not have been referenced yet, so we proceed from Step 7 to Step 8.

In Step 8, the interpreter follows the link in the file data from the entry number written into the operand of the aforesaid instruction and finds the names of the class and the variable (or the method). From the names it has found, it references the various tables and finds the references which specify the locations where the variable and the method program are stored. In Step 9, the interpreter accesses the addresses it obtained by finding the aforesaid references. When it has completed the instruction, it goes from Step 10 back to Step 4. Thereafter, the same processing is executed for each instruction.

When the reference processing is completed in Step 8, the instruction code and operand in the completed instruction are rewritten. When this rewritten instruction is read, the interpreter proceeds from Step 7 to Step 9 and executes the aforesaid instruction without having to find the references.

If we follow the procedure described above, when the first instruction is executed after the system is started, all instructions requiring references must execute the processing in Step 8. In this processing, as was discussed earlier, several stages of referencing must be pursued to arrive at the storage location of the variable or method. This causes a considerable delay between the time the instruction is read and the time it is executed. This causes the processing to be extremely slow when a program is executed immediately after the system is started. Further, since a program's running time immediately after the system is started will be different from the same program's running time the second time or third time it is run, when its references have already been found, it will prove difficult to use such a program in a device for real-time processing which requires an estimated running time.

In prior art systems, even if there were instructions required the same reference in a number of places in a continuous program, that reference had to be found anew for each instruction. This made the referencing efficiency extremely poor.

If the instruction code and operand are rewritten, the object program must be stored in the RAM. It will then take time to load the program in the RAM when the device is started, which will increase the start-up time that the system requires. A RAM with the same capacity as the ROM must be available to store the object program, which drives up the cost of the device.

Summary Of The Invention

This invention was developed in consideration of the problems described above. Its first objective is to realize an information processing scheme such that the necessary references, including reference data specifying locations to be accessed in the memory, are found before the program is executed, and each result is stored in a link to the program. Then the locations to be accessed can be specified immediately by the reference data when the program is exe-

cuted. The program's running time is stabilized and its speed is increased. It is well suited for use in a device which requires real time capability.

The second objective of this invention is to eliminate the need to load the program into a RAM. This would shorten the start-up time for the system, greatly reduce the required capacity of the RAM, and lower the production cost of the device.

The third objective of this invention is to enable the reference results to be read out of the ROM along with the program before the program is executed. This would further reduce the required capacity of the RAM and, when this system is used in a device, enable the program to run at high speed immediately after the system is started up.

With the invention disclosed in Claim 1 of this application, when a program written in an interpreter language is run, the reference data specifying the locations to be accessed in the memory are extracted and the references are found before the program is executed. The results which are obtained are then linked to the program through the aforesaid reference data. When a program is executed which requires that certain data be referenced and accessed in the memory, the locations to be accessed in the aforesaid memory are specified based on the results linked to the program through these reference data.

The invention disclosed in Claim 2 of this application comprises an information processing device which contains a program written in an interpreter language. To enable it to implement the scheme outlined above, it has a means to execute the aforesaid program and a means to link results of referencing to the program through the reference data specifying which locations are to be accessed in the memory. When it is to execute a program requiring that specified data be referenced and accessed in the memory, the means to execute the program uses the results of referencing, now linked to the aforesaid program through the reference data, to specify the locations to be accessed in the aforesaid memory.

If the aforesaid program in an interpreter language is a byte code program compiled from a source program written in Java, it consists of an object program (i.e., an instruction) requiring that names of variables and methods be referenced and data representing the actual contents (the names of the class, the variable, the method, etc.) of the reference data linked to the instruction. The means to execute the program is realized by the function of the interpreter. Through the aforesaid linked data, it verifies the content of the reference data as it executes programs in a given order.

Using the aforesaid reference data to find a reference means referencing certain data by means of the reference data and specifying the locations to be accessed in the memory so that no further reference processing will be necessary.

For example, finding a field reference would mean using the character data indicated by the program, such as the name of the class and variable, to obtain the address where the specified variable is stored and the data which correspond to that address (the aforesaid index to the field table). Finding a method reference would mean using character data such as the name of the method to obtain the address where the specified method is stored and the data which correspond to that address (the aforesaid index to the method table). Finding a class reference would mean using character data representing the name of the class to obtain the address where the data related to the methods and variables in the indicated class (the aforesaid class data) are stored and the data which correspond to that address (the aforesaid index to the class table).

With the invention disclosed in Claim 3 of this application, if the aforesaid program consists of an object program in byte code and data which represent the content of reference data linked to that program, the results of the aforesaid referencing will be stored in the link information of the aforesaid object program. With the invention disclosed in Claim 4 of this application, the link information contains code data of a number of fixed lengths. The aforesaid results of referencing are stored in a location determined by the head code data.

With the invention disclosed in Claim 5 of this invention, the object program in byte code and its link information are read out of the ROM to execute the program.

With the invention disclosed in Claim 1 of this application, the results obtained by ref-
5 erencing the data which specify locations to be accessed in the memory are stored as links to the program before it is executed. This allows the memory to be accessed speedily and the designated processing to be executed immediately after the system is activated. It also stabi-
lizes the program's running time, making it possible to accurately predict that time.

10 The results of referencing the data are linked to the program through the reference data. This makes it unnecessary to rewrite the program as the data are referenced or load the program into the RAM when the system is activated.

15 The fact that the results of referencing are linked to the program without rewriting it means that a program which requires the same reference data to be looked up more than once will not have to repeatedly reference them as was the case in prior art programs.

20 With the invention disclosed in Claim 2 of this application, for every set of reference data the previously obtained results of referencing will be linked to the program at the moment the system is activated. This enables processing to be executed speedily as soon as the program is started up.

25 With the invention disclosed in Claim 3 of this application, the results of referencing the data are stored in the link information indicating the content of the reference data specified by the object program. In this way the results of referencing are linked to the program using the existing configuration of the program's links.

30 With the invention disclosed in Claim 4 of this application, the results of referencing the data are stored in a location specified by the head code data in the link information. This enables the results to be retrieved easily. The location where the referencing results for each

data file are stored is checked before the program is run. This enables the referencing results to be distinguished easily.

With the invention disclosed in Claim 5 of this application, the link information containing the referencing results is stored with the object program in a ROM. This allows the program to run at high speed immediately after the system is started up.

Brief Description Of The Drawings

Figure 1 shows the configuration of an information processing system related to the first preferred embodiment of this invention.

Figure 2 illustrates how the class file data are configured in this embodiment.

Figure 3 shows the order of processing in an information processing system configured as in Figure 1 from the moment the system is started up until the program has been executed.

Figure 4 shows details of the processing in Step 4 of the aforesaid Figure 3.

Figure 5 shows the configuration of an information processing system related to the second preferred embodiment of this invention.

Figure 6 illustrates how the class file data are configured according to a prior art.

Figure 7 shows the configuration of the table of classes and the table of fields used to find field references.

Figure 8 shows the chain of processing in a system according to the prior art.

Detailed Description Of The Invention

Figure 1 shows the configuration of an information processing system related to the first preferred embodiment of this invention.

This information processing system comprises a device which contains a computer. The system executes a byte code program (hereafter simply referred to as "a program") compiled from a Java source program. It consists of storage unit 1 for instructions; storage unit 2 for class file data; unit 3 to execute instructions; unit 4 to manage tables; unit 5 to resolve references; heap memory 6; and unit 7 to assign memory. Of these components, the four processing units, execution unit 3, management unit 4, resolution unit 5 and assignment unit 7, are realized by a Java interpreter. Instruction storage unit 1 is set up in the computer's ROM; data storage unit 2, heap memory 6 and the various reference tables created by management unit 4 are set up in a RAM.

Among the byte code programs, only the aforesaid instructions, i.e., the object program consisting of instruction codes and operands, are stored in the aforesaid instruction storage unit 1.

Class file data consisting of the reference data assembled for each class which indicate the actual content of the references specified by instructions are set up in storage unit 2. The reference data in a given class are represented by numerous data sets linked to the program using the aforesaid entry numbers. The head entry number of the reference data specified by an instruction which requests a reference is written into the operand of that instruction.

Table management unit 4 creates reference tables, such as a class table, a field table and a method table, from the class file data for each class just as was done in prior art schemes. The data organized in these reference tables are supplied as needed to execution unit 3, resolution unit 5 and assignment unit 7.

Execution unit 3 reads the instructions out of storage unit 1 one by one and executes them.

5 Memory assignment unit 7 sets up a work area in heap memory 6 for each instance generated by the execution of the program in response to a command from execution unit 3. Based on the value of the aforesaid object offset, it assigns a given address in the said work area to the variables for each instance.

10 Resolution unit 5 resolves references requested by an instruction. In this embodiment, references indicated by name data contained in the class file are resolved by finding the names before the program is executed, and the results obtained are written into the class file data.

15 The technique used to resolve the references is identical to that employed in the prior art. For example, the result of resolving a field reference would be a field table index, and the result of resolving a method reference would be a method table index. These indices would be written into the class file data. For a class reference, a class table index would be obtained for the class name that was indicated and stored as a result in the class file data.

20 Figure 2 illustrates how the class file data are configured in this embodiment. Just as in the prior art configuration shown in Figure 6, the figure extracts the reference data for a single instruction.

25 In this embodiment, as in the prior art, various code data are linked to the code data for entry number 6, the "field reference" which is at the head of the list. These character data indicate the name of the class, variable and type of variable, as well as what sort of reference they represent.

These code data have a fixed length. As in the prior art, a flag is stored at the head of each set of code data. Other data are stored at the end of the code (in the example, on the right side). A blank space (indicated in the drawing by a dash) is left in the center.

5 Because this embodiment requires that all code data must have a fixed length, the character string representing the name is stored in a separate table of character strings. The aforesaid table is organized in class units. The table number for the character string table (in the drawing, 0) and the locations of those character strings in the table (in the drawing, represented by the numerals 1000, 1004 and 1010) are stored as location data in entry numbers 17,
10 20 and 27.

The aforesaid results of resolving the references are stored in the blank space provided in the head entry of the aforesaid reference data. In the example shown, the field table index obtained by resolving the field reference (hereafter referred to as the "resolved field table index") is stored in the blank space in entry number 6, which is at the head of the link for the aforesaid reference data. The class table index obtained by the process of resolving the aforesaid field reference (hereafter referred to as the "resolved class table index") is stored in the blank space in entry number 1, which is linked from entry number 6 to the storage location of the class name. The blank spaces in entry numbers other than 1 and 6 remain empty.

20 The location in which the index is stored in the aforesaid blank space is fixed, so it is easy to determine whether a field reference can be resolved by looking at the variable name.

25 For reference data which represent a method name in the class file data, the method table index obtained by resolving the method reference (hereafter, the "resolved method table index") is written into the head entry of the aforesaid reference data in the same way.

Figure 3 shows the order of processing in an information processing system configured as in Figure 1 from the moment the system is started up until the program has been executed.

In Steps 1 through 3 in the figure, when class file data have been created for a designated class, various reference tables, including class, field and method tables, are set up just as in the prior art.

5 In Steps 4 through 6, the aforesaid resolution unit 5 resolves the references using the variable and method names in the class file data. Until references have been resolved for all the class file data, classes whose references are still unresolved will be designated in order in Step 6, and their references will be resolved. When references have been resolved with respect to the class file data for every class, the answer in Step 5 will go to "no", and instructions will be executed in order in the loop in Steps 7 and 8.

10 Figure 4 shows details of the processing in Step 4 of the aforesaid Figure 3. To simplify the explanation, the order shown is that used to obtain a field table index by resolving a field reference.

15 We shall next explain, with reference to the flow chart in Figure 4, the order of processing used to resolve the field reference in the class file data shown in the aforesaid Figure 2.

20 In Step 4-1, resolution unit 5 extracts from the class file data an entry (entry number 6) whose references have not yet been resolved. This will be an entry with the field reference tag "09" and which does not have a resolved field index. In Step 4-2, unit 5 traces in order, from the aforesaid data it extracted in Step 4-1, the data for the class name reference (entry number 1) and its link information (entry number 20), to obtain the name of the class to which the aforesaid variable belongs.

25 Resolution unit 5 checks whether a class with this name can be found in the class table. If it is there, unit 5 obtains a class table index for that class (Steps 4-3 and 4-5). If there is no class in the table with that name, in Step 4-4 unit 5 uses the aforesaid class file data for that class to enter it in the class table and the field table, and it obtains a class table index.

From the data in entry number 6 it extracted in the aforesaid Step 4-1, resolution unit 5 then follows, in order, the reference data for the name and type (entry number 9) and their link information (entry numbers 27 and 17). In this way it obtains the variable and class names (Step 4-6). It retrieves the field table and gives it the class table index obtained in the aforesaid Step 4-5. It extracts the field data which match the variable and type names obtained in Step 4-6 and obtains the field table index attached to these data (Step 4-8).

When it obtains the field table and class table indices in the aforesaid Steps 4-5 and 4-8, resolution unit 5 writes them into the designated blank space in the aforesaid class file data (Step 4-9).

Unit 5 continues, in the same way, to extract in order the data whose references are not yet resolved, the data which have the field reference tag "09". It obtains field and class table indices from the class, variable and type names it finds by following the data linked to these, and it writes the values for each index in the class file data.

When unit 5 retrieves a field table and finds that it has no field data which match the combination of class table index, variable name and type name, the answer in Step 4-7 goes to "no", and we move to Step 4-11, which is processed as a link error.

Thus the field references for every program variable are resolved through the variables themselves before the program is executed. The resolved field table index is stored in the blank space within the entry number representing the instruction which indicates that field reference. Thus when execution unit 3 is going to execute the aforesaid instruction 10, it obtains the aforesaid resolved field table index from the entry number written into the operand of instruction 10. It accesses the field it finds in the object offset value it obtains from the values in the index and executes the instruction.

The reference data containing character data representing the name of the class (consisting of entry number 1 with the aforesaid tag "07" attached to it and entry number 20, the

linked entry number which represents the character string for the class name) are linked to the instruction which required the class reference. The aforesaid resolved class table index is written into the blank space in the head entry number. Accordingly, execution unit 3 obtains the aforesaid resolved class table index through the entry number written into the operand. By
5 reading out the class data which correspond to this index, it can easily obtain the data associated with the objects and methods in that class.

References are also resolved in the same way before executing the program for instructions requiring that a method be referenced. In this case a resolved method table index is
10 written into the head entry number of the reference data linked to the instruction. When this instruction is to be executed, the method table index obtained from the entry number in the instruction's operand tells unit 3 where the program for the method is stored, and the method can be accessed immediately.

If a program contains an instruction which generates a string object representing a character string in the class, the references are resolved for that object's name the first time the instruction is executed. The results are written into the class file data so that instructions which require that the string object be referenced can be executed at high speed.
5

Two tables are set up for a string object: a character string table in which is stored a character string representing that object and an object table which indicates the class to which the object belongs and the location in the aforesaid character string table where its character string is stored. When an instance is generated, data are entered which are linked mutually and reciprocally to the aforesaid character string and object tables. When a reference is resolved,
20 just as was described above, an object table index obtained by resolving the references is written into the head entry number of the reference data linked to the instruction. Thereafter, whenever a program requires that a string object be referenced, the object table index for the object indicated by the class file data is read out. The location in the character string table where the string is stored can be found quickly so that the pertinent character string data can
25 be accessed.
30

With the information processing system described above, when name data are referenced, all the reference data are extracted from the class file and the reference processing is resolved before the instruction indicating that a given address should be accessed is carried out.

5 Since the results are stored in the location where read-out of the reference data linked to the instruction's operand begins, the addresses of the destinations to be accessed for each instruction can be completely specified using the results in the class file data as soon as the program is started up, and the instructions can be carried out at high speed. There will be no difference between the program's running times the first time it is executed and the subsequent times it

10 is executed. This allows the program's running time to be predicted accurately.

Since the results of referencing are written into the aforesaid head entry of the operand without rewriting the instruction, the results are linked to the program using the existing link configuration. To execute an instruction using these referencing results, execution unit 3 has

5 only to read the results of referencing out of the head entry written in the operand. There is no need to significantly change the design of the interpreter.

Because the referencing results are written in a specific location in the head entry, they can be read out easily. When resolving the references, the processing involved in finding data

20 whose references are still unresolved is also simplified.

In the information processing system of this embodiment, there is no need to rewrite instructions. Reading an instruction out of the ROM and executing it doesn't require that the program be loaded at start-up time. Once power is supplied to the device, the system can start

25 up swiftly. Because it is not necessary to load the program, the capacity of the RAM can be substantially reduced.

Because the aforesaid results of referencing are stored so as to correspond to the reference data in the class file, instructions which require that the same data be referenced can share

the results. There is thus no need to resolve the same reference repeatedly, and the references can be resolved more efficiently.

To apply the method of storing the referencing results for each set of reference data in the aforesaid class file data, the class file data in which are written the referencing results already obtained can be stored in the ROM, or a reference table can be created according to the class file data and both the data and the table can be stored in the ROM. With this method, it will not be necessary to resolve the references after starting up the system. As soon as the program is activated, high-speed processing can commence. This information processing system is particularly suited for use in compact devices which require real-time processing.

Figure 5 shows the configuration of an information processing system related to the second preferred embodiment of this invention.

The information processing system of this embodiment has the same configuration as that shown in Figure 1 above, with the exception of resolution unit 5. However, all the reference tables created by storage unit 2 and table management unit 4 are set up in the ROM along with instruction storage unit 1. The rest of the configuration is identical to that of the aforesaid first embodiment.

Class file data for all classes stored in instruction storage unit 1 are stored in storage unit 2 for class file data. Table management unit 4 manages the class, field, method and character string tables in which class data are recorded.

Index data representing the results already obtained by resolving the references for the variable, method and class name for each class are stored in storage unit 2 for class file data, just as in the aforesaid first embodiment.

With an information processing system configured as described above, the system is activated as soon as power is connected. Class file data in which the referencing results are writ-

ten and reference tables are set up, and all instructions are carried out at high speed based on the results of referencing those instructions. This information processing system does not require a large-capacity RAM, so the device which contains it can perform high-speed information processing at a low cost.

5

With the inventions disclosed in Claims 1 and 2 of this application, the reference processing is resolved before the program is executed using the reference data which specify the locations in the memory to be accessed. The results of referencing are then linked to the program through the reference data. Thus when the program is started up, the stored referencing results can be used to access the memory quickly. This scheme stabilizes the program's running time and allows it to run at high speed. Because the program's running time is invariant, the information processing system is suitable for a device using real-time processing, which requires an accurate prediction of the program's running time.

10

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197

written into the aforesaid designated location. This allows data with unresolved references to be recognized easily so that resolution processing can be speeded up.

With the invention disclosed in Claim 5 of this application, the program to be executed
5 and the referencing results linked to it are read out of the ROM before the program is run. Then as soon as power is connected, the program can be activated and high-speed processing can be performed. There is no need to load the program or the results into the RAM, so the RAM's capacity can be further reduced. This scheme allows a device to have a program in it which can execute high-speed processing at a low cost.